

Decision Making 1

Noel Welsh

04 November 2010

1 Motivation

Last lecture we saw that the behavioural robotics provides a good framework for constructing low level and intermediate level behaviours for a mobile robot.

However, the methodology of behavioural robotics is lacking. There is no route to creating and organising behaviours beyond “extensive real world testing”. This is extremely time consuming and doesn’t scale. If we are to create truly intelligent robots they must be able to learn behaviours. This is what we (start to) look at today.

2 The Problem Setup

There are many problems we could consider, such as constructing behaviours from low-level primitives, or learning hierarchies of control. Today we’re going to consider a relatively simple task, that of choosing between a fixed number of pre-defined behaviours. This a reasonable thing to do as we’ve seen we can relatively easily create low-level behaviours.

For example, if our robot is an insect-like device we might equip it with several foraging behaviours. We might then want it to learn the best foraging behaviour via its interactions with the environment.

There are two important restrictions that we will use: we assume that the action choice makes no use of observations, and that actions do not significantly affect the world. The first restriction we can get around in practice by having, say, a subsumption architecture that decides on the state of the robot (e.g. foraging, evading predator). Given this state we can then choose between the actions using the algorithms we’re going to consider. The second restriction means that the distribution covering the outcome of an action does not change over time. This does *not* mean the outcome of an action is always the same. This is a reasonable restriction for an insect-like creature to make, as it makes little permanent impact on the world. If we have time we’ll see in later lectures how these restrictions can be lifted.

This problem setup is known as a *multi-armed bandit*. The name comes from the “one-armed bandit” also known as fruit, slot, or poker machines. The one-armed bandit has a single arm to pull. In the problem we are considering the machine has multiple arms. We want to learn which arm gives the best rewards over time.

The multi-armed bandit problem has important applications outside of robotics. For example, choosing the best advertisement to display to a web user is a multi-armed bandit problem. This task was worth approximately *23 billion* dollars to Google in 2009. Choosing between different drugs to treat a patient is also a multi-armed bandit problem. You get the idea.

3 No Regrets

We need to formalise our measure of success a bit before we can examine different algorithms for the bandit problem. We'll add additional notation as we need it.

Everyone should know the world is made of win and fail. We want to create an algorithm that wins as much as possible. This means choosing the best action. Each time we choose an action we get a *reward*, which for convenience we say is a number between 1 and 0. The higher the reward the higher our win. Rather than maximising reward it is easier to minimise our *regret*, the amount we fail compared the best possible action.

When developing algorithms we want to consider performance over all possible sequences of actions, rather than any specific sequence we might observe. So we are going to look at average rewards and hence average (or expected) regret. We denote the average (or mean or expected) reward of the best action as μ^* and of any other action j as μ_j . There are a total of K actions. We write $T_j(n)$ for the number of times we have tried action j in a total of n action. Formally, the regret after n actions is defined as

$$regret(n) = \mu^*n - \sum_{j=1}^K \mathbb{E}[T_j(n)] \quad (1)$$

Note that \mathbb{E} is the symbol for expectation (or mean or average). Thus the regret calculates the average fail given the reward we expect to get from the different actions, and the number of times we expect to play sub-optimal actions.

4 Hypothesis Testing

One approach to the bandit problem is to frame it as a hypothesis testing problem. We could try every action an equal number of times and, after a fixed number of actions, use a hypothesis test to determine the best action. There are a number of issues with this:

- We make no use of the information we gain during the trial. Thus our regret will be higher than it need be.
- We may not be able to see a statistically significant result. Then what do we do?
- Most hypothesis tests are setup to minimise Type 1 errors (falsely rejecting H_0) at the expense of increasing Type 2 errors (falsely accepting H_0). This doesn't seem a sensible thing to do in this case. We can adjust the probability of the

different errors by changing the p-value, but as we decrease one probability we increase the other. The essential problem here is we are forcing ourselves to make a hard cut-off when really we can keep trying different actions indefinitely.

So hypothesis testing is not the correct framework for the bandit problem. Despite this, hypothesis testing is used in practice for decision making problems such as the aforementioned medical problem, and under the term “A/B testing” on websites. You might want to ponder why this is. (I can’t come up with a *good* reason.)

5 Exploring and exploiting

An alternative we might try is to choose the action with the highest average reward so far. This seems like it would work but can give regret the scales linearly with the number of plays.

Imagine we have a two actions, both of which give rewards according to a Bernoulli (binary) distribution. Imagine the first time we try action 1 is gives a reward of 0. The first time we try action 2 is gives a reward of 1. Now the average reward of action 2 will never drop to zero so we will never try action 1 again.

This illustrates a classic problem which is the defining characteristic of *decision making*: the trade-off between exploring and exploiting. Exploring means to try new actions to learn their effects. Exploiting means to try what we know how worked in the past. If we exploit too much, as in the example above, we will never find better ways of accomplishing our task. If we explore too much we will never make use of what we’ve learned in the past.

6 ϵ -greedy

A simple modification to the above algorithm is to choose the best action with a probability $1 - \epsilon$, and to otherwise choose an action at random. This is known as ϵ -greedy. It is straightforward to see that if we keep ϵ constant the regret will grow linearly with the number of actions. A better way is to use some function $\epsilon(t)$ that decreases over time. This makes our algorithm shift from exploration to exploitation as our estimates of the rewards improve. If we still play each arm infinitely often (so $\sum_{t \geq 1} \epsilon(t)$ diverges) then our average reward will converge to the true mean. With this setup we can converge to the best action. See [?] for details.

7 Optimism FTW

The ϵ -greedy algorithm could still be improved. As given it explores uniformly over all actions. Better would be to use our estimates of reward to guide exploration, so we try actions that appear better more often. So long as we still try every action infinitely often in the limit we know that our estimate of reward will converge.

One way to do this is to try each action with a probability proportional to our current estimate of reward. However we can do better than this! If we have tried an action less

often then our estimated reward is less accurate. One approach is to add on a certain amount to each reward, giving an upper bound on where we expect the true average reward to lie. The amount we add should decrease with the number of times we have tried an action. This is called an *optimistic* policy; we assume each action is good until evidence shows it isn't.

How do we know how much to add? We can turn to the classic Chernoff-Hoeffding bound to get (most of the way to) an answer. The Chernoff-Hoeffding bound says:

Let X_1, X_2, \dots, X_n be independent random variables in the range $[0, 1]$ with $\mathbb{E}[X_i] = \mu$. Then for $a > 0$,

$$P\left(\frac{1}{n} \sum_{i=1}^n X_i \geq \mu + a\right) \leq e^{-2a^2n} \quad (2)$$

and

$$P\left(\frac{1}{n} \sum_{i=1}^n X_i \leq \mu - a\right) \leq e^{-2a^2n} \quad (3)$$

If we wanted to put an upper bound of p on the average reward, we can solve $p = e^{-2a^2n}$ for a to find out how much we should add. Try this. You should get $a = \sqrt{-\frac{\ln p}{2n}}$.

8 UCB1

The algorithm UCB1 (for *upper confidence bound*) is an algorithm for the multi-armed bandit that achieves regret that grows only logarithmically with the number of actions made. It is also dead-simple to implement. The algorithm works as follows:

- For each action j record the average reward \bar{x}_j and number of times we have tried it n_j . We write n for total number of actions we have tried.
- Try the action that maximises $\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$

That is all! From our analysis of the Chernoff-Hoeffding bound above we can see that the confidence bound grows with the total number of actions we have taken but shrinks with the number of times we have tried this particular action. This ensures each action is tried infinitely often but still balances exploration and exploitation.

The regret for UCB1 grows at a rate of $\ln n$. In particular, after n actions it is at most

$$\sum_{j=1}^K \frac{4 \ln n}{\Delta_j} + \left(1 + \frac{\pi^2}{3}\right) \Delta_j \quad (4)$$

where $\Delta_j = \mu^* - \mu_j$.

9 Notes

These notes also cover most of the same material as us: <http://www.cs.caltech.edu/courses/cs253/slides/cs253-07-ucb1.pdf>

Fail Road by Firefly the Great (Dagny Scott) <http://www.flickr.com/photos/fireflythegreat/2845637227/>

Lynx High Five by Amadeus Varadi Hellequin <http://www.flickr.com/photos/rucken/4411674785>

10 Bibliography